INTERFACE METHOD AND DEVICE HAVING INTERFACE FOR CIRCUIT COMPRISING LOGICAL OPERATION ELEMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention:

The present invention relates to an interface method and device having an interface for use with a logical circuit which comprises a logical operation element including a flip-flap and a variety of gates such as an AND gate, an OR gate or a NOT gate.

2. Description of the Related Art:

According to a conventional interface method employed for interfacing a microprocessor and a peripheral LSI, a status value is read out and/or execution of predetermined processing is controlled via a register referred to as a status register or a command register. Therefore, programmers who design a program for controlling the peripheral LSI must be familiar with the meanings of and correlation between the respective registers in order to understand the interface of the peripheral LSI, and are required to mind the procedure of setting the registers in preparation of programs.

Recent peripheral LSIs can accommodate more complicated functions, and accordingly are equipped with a larger number of status and/or command registers. This makes it difficult for programmers to become familiar with the meanings of or correlation between the respective registers. This difficulty potentially increases program errors and causes a longer error determination time. Moreover, due to the complexity of functions, determination

can be hardly made as to whether such a function is realized by means of software or hardware. Such a background leads to a need of a method and/or device which can facilitate exchange between software and hardware.

Meanwhile, according to one method that has recently become common, an interface with a software object is defined using IDL(interface definition language). Examples of such a language may include an IDL of CORBA by OMG(Object Management Group) and an IDL by COM(Component Object Model) by Microsoft. These languages, however, are directed to only software objects and cannot accommodate hardware objects, or a device.

In view of the above, the applicant of the present application conceived an idea that, if an interface with a hardware can be defined using an IDL, hardware and software could be treated uniformly without discrimination. This could enable, for example, ready exchanges between software and hardware, thus reducing the burden on a programmer. Moreover, this could also enable, for example, ready definition of a method for interfacing with a software, thus also reducing the burden on a hardware designer.

The present invention has been conceived in view of the above and advantageously offers an interface method for a logical circuit and a device having an interface. According to the method or device of the present invention, interfacing with a device, such as a peripheral LSI, which comprises a logical circuit can be improved and the burden on a programmer in understanding registers and their correlation can be reduced. Moreover, the method or device of the present invention allows uniform treatment of a software object and a hardware object, thus enabling ready exchange between these objects.

SUMMARY OF THE INVENTION

In order to produce the above described advantages, according to one aspect of the present invention, there is provided (1).

According to another aspect of the present invention, there is provided (2).

According to still another aspect of the present invention, there is provided (3).

According to yet another aspect of the present invention, there is provided (4).

According to yet another aspect of the present invention, there is provided (5).

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram showing a structure of a computer system of the present invention;

Fug. 2 is a block diagram showing a structure of a random number generator;

Fig. 3 is a diagram explaining an interface definition of a random number generator using an interface definition language, or an IDL;

Fig. 4 is a diagram explaining a header file corresponding to a program of Fig. 5;

Fig. 5 is a diagram explaining a program for directly causing a random number generator to operate;

Fig. 6 is a flowchart of an operation of an IDL server interface section;

Fig. 7 is a flowchart of a designing operation;

Fig. 8 is a block diagram showing a computer system incorporating a pi calculator;

Fig. 9 is a diagram explaining an interface definition for use by a pi calculator;

Fig. 10 is a block diagram showing a structure of a pi calculator which calculates a pi value;

Fig. 11 is a block diagram showing a structure of a computer system modifies so as not to use a random number generator;

Fig. 12 is a flowchart of request monitoring and function execution by a CPU;

Fig. 13 is a diagram explaining a header file corresponding to a program of Fig. 14;

Fig. 14 is a diagram explaining a program describing the process of Fig. 12 in C-language; and

Fig. 15 is a flowchart of an operation of an IDL client interface section.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following, embodiments of the present invention will be described with reference to the accompanied drawings.

Fig. 1 is a diagram showing a structure of a computer system of the present invention. This system comprises a CPU 1, a memory 2, and a random number generator 3. The memory 2 and the random number generator 3 are connected to the CPU 1 via a system bus 4. The CPU 1 interprets and processes a program stored in the memory 2.

Fig. 2 is a block diagram showing a structure of the random

number generator 3. The random number generator 3 has an IDL server interface section 11, which comprises a function identifying register 12, an argument register 13, a return value register 14, and a status register 15. The IDL server interface section 11 is connected to the system bus 4 via an external terminal 16. It should be noted that although a controller for controlling data exchange between the system bus 4 and the respective registers 12 through 15 is not shown in Fig. 2, the data is exchanged via an address bus or a control bus contained in the system bus 4.

Fig. 3 shows an example of a definition of an interface written in an IDL. This definition, as is common to an IDL of CORBA described above, specifies an interface name, a function name, and an argument and a return value for each function. The example of Fig. 3 defines an interface called "randomGenerator" of a random number generator 3, and specifies a function for setting a random seed (function name: setSeed) and a function for obtaining a generated random number (function name: get Random). Function "setSeed" is for supplying a seed in a double precision real number as an input argument, and function "getRandom" is for returning a random number generated as a return value, in the form of a double precision real number.

Figs. 4 and 5 respectively show a header file and a program written in C-language based on the interface definition of Fig. 3 to operate a device having an interface of the present invention by the CPU 1. Generally, such a header file and a program can be created using a program known as an "IDL compiler" by converting an interface definition according to a given rule. While an IDL compiler such as CORBA converts the interface definition into a header file or a program written in C or C++ language to creates

communication data based on the interface definition, in this embodiment, a program to set or refer to registers of the IDL server interface section 11 of the random number generator 3 written in C-language is generated. Such a program is generally referred to as a stub or a proxi.

The program of Fig. 5 specifies functions "setSeed" and "getRandom" respectively corresponding to "setSeed" and "getRandom" included in the interface definition. These functions will next be described in the context of an operation of the random number generator 3 while referring to the structure of Figs. 1 and 2, as the method and device of the present invention are achieved through cooperation by two or more sections.

Specifically, the CPU 1 reads out a program stored in the memory 2, and then calls a function "setSeed" from the read program. Based on a program relevant to the function "setSeed", the CPU 1 writes "FID_setSeed" which is a value to identify functions into the function identifying register 12 of the random number generator 3. The CPU 1 also writes an argument "seed" into the argument register 13 as a double precision real number, to thereby complete the processing. It should be noted that that the constant "FID_setSeed" is defined by the header file of Fig. 4, as other constants such as "FID_getRandom" and "Finished" are similarly defined by a header file.

Then, the IDL server interface section 11 of the random number generator 3 confirms receipt of all arguments relevant to the function "setSeed" and immediate thereafter begins its operation. In this example, it stores a random seed in the random seed register 18 of the random number generating section 17.

Subsequently, function "getRandom" is called, and a

function identification value "FID_getRandom" is written into the function identifying register 12 of the random number generator 3. Because the function "getRandom" contains no argument, the IDL server interface section 11 immediately causes the random number generating section 17 to generate a random number, and writes the generated random number into the return value register 14.

During processing for the function "getRandom", a value in the status register 15 is continuously read outby calling function "getStatus" until any value is written into the return value register 14 and the processing for the function "getRandom" is repeated until a value corresponding to "Finished" is written in the return value register 14.

The above is an outline of the overall operation of a computer system incorporating a device of the present invention. The computer system shown in Fig. 1 is designed as explained below and shown in the flowchart of Fig. 7, in which each step is represented as "S" followed by a corresponding number.

Initially, an interface is determined (S21) by listing all functions necessary for the interface, and specifying a function name, an argument, and a return value for each function. Then, an interface definition file is created based on the interface defined at S21 (S22), and the resultant interface definition file is supplied to an IDL compiler to create a header file, such as is shown in Fig. 4, and a stub, such as is shown in Fig. 5 (S23).

The operation flow then branches into software designing and hardware designing. For software designing, the stub is complied by a compiler to thereby create a relocatable object (S24). An execution module is created by linking the resultant relocatable

object to relocatable objects of different functions (S25). Then, the software design process is completed.

For hardware designing, on the other hand, a hardware designer designs an IDL server interface section 11 with reference to the interface definition file of Fig. 3 and the header file of Fig. 4 (S26). The content of the interface definition file and header file to be referred to here will be described below in detail in the context of an operation of the IDL server interface section 11. Then, a circuit for connecting a circuit of the random number generating section 17 and the IDL server interface section 11 is designed (S27) to thereby complete designing of the random number generating device 3. Thereafter, a circuit for connecting the CPU 1, the memory 2, the random number generator 3, and the system bus 4 are designed (S28) to thereby complete hardware designing.

Functions of the IDL server interface section 11 will next be described in detail with reference to the flowchart of Fig. 6 as these are essence of the present invention.

Specifically, the IDL server interface section 11 determines whether or not any value is written into the function identifying register 12 via an external terminal 16 (S1). A value to be written into the function identifying register 12 may be, for function "getSeed", a value indicative of "FID_setSeed" as defined by "#define" in the header file of Fig. 4, or "1" here. For function "getRandom", on the other hand, a value indicative of "FID_getRandom", or "2" here, may be written.

With any value written into the function identifying register 12, the value of the status register 15 is changed to a value "Executing", which means "in process" (S2), and the number of arguments is thereafter found out with reference to the content

of the function identifying register 12 (S3) by an argument number detection section 20. The argument number detection section 20 is prepared in advance, with reference to the interface definition file of Fig. 3 and the header file of Fig. 4, so as to store correlation between a function identification and the number of relevant arguments.

Then, whether or not all arguments have been written into the argument register 13 is determined (S4). If there exists any argument which has yet to be written, the operation repeats from S4 (S5). If all arguments have already been written, or when there originally exists no argument, the function specified by the function identification is initiated (S6). A designer of the IDL server interface section makes a function identification coincident with a corresponding function, with reference to the header file of Fig. 4.

When processing relative to the function is completed (S7), whether or not there is any return value is determined (S8). If there is, the return value is written into the return value register 14 (S9). After finally changing the value of the status register 15 to "Finished", the above-described processing at S1 and subsequent steps is repeated.

It should be noted that, in the program of Fig. 5, although a "while" loop is used for waiting until a value of the status register is changed to "Finished", the interrupt which is caused by changing the value of the status register to "Finished" may be used for waiting. This arrangement can improve efficiency of use of the CPU 1.

The example of the embodiment of the present invention described above can be modified such that the number of arguments

and/or the type of data are additionally determined. Also, in the case where no function in an interface definition contains an argument, the argument register 13, and thus S4 and S5 in Fig. 6, can be omitted. When no function contained in an interface definition returns a return value, the return value register 16, and thus S8 and S9 in Fig. 6, can be omitted.

Emulation of the operation of the random number generator 3 by software can be achieved merely by replacing the functions"setSeed" and "getRandom" of Fig. 5, which are hardware objects, by corresponding software objects. This is easily achievable as a hardware object can be readily replaced by a software object according to the present invention because an interface definition language used in the method and device of the present invention is partly common to an interface definition language directed to a software object.

Fig. 10 shows a structure of a pi calculator 5 for calculating a circuit rate, or a pi, using a random number generator 3 of Fig. 2 according to a Monte Carlo method. Fig. 9 shows an interface definition of the pi calculator. Fig. 8 shows a structure of a computer system incorporating the pi calculator 5.

The pi calculator 5 of Fig. 10 comprises an IDL server interface section 21, a pi calculating section 27, and an IDL client interface section 31. The IDL server interface section 21 is connected to the system bus 4 via an external terminal 26. The IDL client interface section 31 is connected to an external terminal 16 of the random number generator 3 via an external terminal 36.

Connection between the external terminal 36 and the external terminal 16 enables data transfer from the function identifying register 32 to the function identifying register 12, between the

argument register 33 and the argument register 13, and from the return value register 14 to the return value register 34, whereby the pi calculator 5 can access the random number generator 3.

Fig. 11 shows a structure of the computer system of Fig. 8 which is modified so as not to use the random number generator 3. In the structure of Fig. 11, the IDL client interface section 31 of the pi calculator 5 is connected to the system bus 4 via the external terminal 36. In order to modify the structure of Fig. 8 to that of Fig. 11, the IDL client interface section 31 of the pi calculator 5 should be able to operate the IDL server interface section 11 of the random number generator 3, and values can be read and written via the CPU 1 with respect to the function identifying register 32, the argument register 33, the return value register 34, and the status register 35.

With the structure of Fig. 11, the CPU 1 should monitor a request from the IDL client interface section 31 of the pi calculator 5 and apply a function specified by a function identification.

Fig. 12 is a flowchart of request monitoring and function execution by the CPU 1. Specifically, a value of the status register 35 of the IDL client interface section 31 of the pi calculator 5 is read out (S31). When the read value indicates "Requesting", it is determined that function execution has been requested(S32), and a value of the function identifying register 32 is then read out (S33). When the read value of the function identifying register 32 indicates "FID_setSeed" (S34), a value of the argument register 33 is further read out and stored as a random seed (S35). When the read value of the function identifying register 32 indicates "FID_getRando" (S36), a random number is generated (S37) and is written into the return value register 34 (S38). After S35 or S38,

the operation flow returns to S31.

Execution of this program exclusively occupies the CPU 1. Therefore, the efficiency of the CPU1 may be improved by including an interrupt when the value of the status register 35 is changed to "Requesting".

Figs. 13 and 14 respectively show a header file and a program both describing the process of Fig. 12 in C-language. The header file and the program, which can be uniquely obtained through conversion from an interface definition according to a given rule, can be created using a program called an IDL compiler, similar to the header file and a program file of Figs. 4 and 5. A program such as is shown in Fig. 14 is referred to as a skeleton or a stub.

Fig. 15 is a flowchart of an operation of an IDL client interface section 31 having the above-described property.

Initially, whether or not the pi calculating section 27 has requested execution of a function is determined (S41). When it has, a corresponding function identification is written into the function identifying register 32 (S42). Specifically, a value indicative of either "FID_setSeed" or a "FID_getRandom" is written here. Then, whether or not the requested function is "setSeed" is determined (S43). If it is determined that the requested function is "setSeed", an argument is received from the pi calculator 27 and written into the argument register 33 (S44) before the value of the status register 35 is changed to "Requesting" (S45).

Hereafter, examples will be described in which different operations are executed according to whether or not the external terminal 36 of the pi calculator 5 is connected to the random number generator 3 or CPU 1.

Specifically, when the external terminal 36 is connected to

the CPU 1, the CPU 1, which at this point must be operating according to the flowchart Fig. 12, reads out an argument from the argument register 33 at S35 with respect to the function identification "FID_setSeed". With respect to the function identification "FID_getRandom, there is no argument. Whether or not the CPU 11 has read out all arguments is determined (S47). A case where there originally exists no argument is treated as a case where all arguments have been read.

When all arguments have been read, it is then determined whether or not there exists any return value (S48). When it is determined that one or more return values exist, a return value is written into the return value register 34, and whether or not all return values have been written into the return value register 34 is determined (S49). When it is determined that all return values have been written into the return value register 34, the value written in the return value register 34 is supplied to the pi calculating section 27 as a random number (S50). This is the end of the processing involving the CPU 1, and the value of the status register 35 is changed to "waiting" (S51) before the operation flow returning to S41 to repeat the above.

Returning to S46, when the external terminal 36 is connected to the random number generator 3, the content of the function identifying register 32 is written into the function identifying register 12 of the IDL server interface section 11 of the random number generator 3 (S52). The random number generator 3 then must operate according the flowchart of Fig. 6 and, upon detection at S1 (Fig. 6) of the fact that data has been written into the function identifying register 12, applies processing at S2 and thereafter. It should be noted that that processing at S53 (Fig. 15), and

thereafter by the IDL client interface section 31, is carried out in association with processing at S4 (Fig. 6), and thereafter by the IDL server interface section 11 of the random number generator 3.

Returning to Fig. 15, whether or not there exists any argument is determined (S53). Determination of whether or not there an argument exists is made in response to a function identification value indicative of "FID_setSeed". In this case, the content of the argument register 33 is written into the argument register 13 of the argument register 33 (S54).

Then, whether or not there is any return value is determined (S55). It is determined that there is a return value in response to a function identification value indictive of "FID_getRandom". When there is any return value, whether or not the value of the status register 15 of the random number generator 3 is "Finished" is determined (S56). When it is determined that the random number generator 3 is "Finished" , a value of the return value register 14 of the random number generator 3 is written into the return value register 34 (S57), and the content of the return value register 34 is supplied to the pi calculating section 27 (S58). This is the end of the processing involving the random number generator 3; the value of the status register 35 is changed to "waiting" (S51) and the operation flow returns to S41 to repeat as necessary the process described above.

It should be noted that, although the present invention has been described while referring to the random number generator 3 and the pi calculator 5 as an example in the above embodiment, the present invention can be applied to various types of devices in which one or more of the random number generating section 17, the

pi calculating section 27, and the interface definition (randomGenerator, piCalculator, etc.), are modified.

It should also be noted that, although the type and/or number of arguments is not ascertained in the above examples, such information may be determined in order to achieve more strict detection of errors in an interface program.

It should further be noted that, although only an example of an input argument is described in the above embodiments, an output argument and/or an input/output argument can be similarly applied, although, for an output argument, the direction of data transfer is reversed relative to the situation with an input argument.

It should still further be noted that, although an IDL of CORBA is referred to an interface describing language in the above embodiments, the interface describing language is not limited to an IDL of CORBA and any language which is at least partly common in an IDL directed to a software object and has a means for defining types of arguments and/or return values used in respective functions can be employed.

It should also be noted that, although discrimination of an instant object contained in the simplified program is not described in the above examples, the program preferably has a means for discriminating instant objects.

It should yet further be noted that, although the method and the device of the present invention can be most effectively used for interfacing between an integrated circuit and an external device or unit, the present invention can be similarly applied to interfacing between blocks within an integrated circuit or between logical circuits comprising two or more integrated circuits.

According to the method and the device of the present invention,

interface with hardware can be defined using an interface definition language which is partially common to a software object. This allows uniform treatment of software and hardware, thus enabling ready exchanges between a software object and a hardware object, or a device. As a result, the burden on a programmer can be significantly reduced.

Further, because a method for interfacing with a software can be more easily defined, the burden on a hardware designer can also be remarkably reduced.

Still further, because hardware and software can be developed in parallel, the time required for development can be shortened.

Yet further, because connection between hardware objects and with a CPU can be easily exchanged , hardware processing and software processing can be easily substituted.